An Efficient Algorithm for a Sharp Approximation of Universally Quantified Inequalities

A. Goldsztejn
CNRS, University of Nantes
France
alexandre@goldsztejn.com

C. Michel
University of Nice
France
cpjm@essi.fr

M. Rueher
University of Nice
France
rueher@essi.fr

ABSTRACT

This paper introduces a new algorithm for solving a subclass of quantified constraint satisfaction problems (QCSP) where existential quantifiers precede universally quantified inequalities on continuous domains. This class of QCSPs has numerous applications in engineering and design. We propose here a new generic branch and prune algorithm for solving such continuous QCSPs. Standard pruning operators and solution identification operators are specialized for universally quantified inequalities. Special rules are also proposed for handling the parameters of the constraints. First experimentation show that our algorithm outperforms the state of the art methods.

Categories and Subject Descriptors

G.1 [**Numerical Analysis**]: Miscellaneous

General Terms

Theory, Algorithms

Keywords

Quantified constraints, continuous domains, interval arithmetic

1. INTRODUCTION

Many applications in engineering require verifying safety or performance conditions on a given system. For instance verifying for all electrical current in the interval $[i_{\min}, i_{\max}]$ and all possible resistor values in the interval $r_0 \pm 5\%$ that the voltage across the resistor remains lower than a safety bound $u_{\max}$. In other words, we have to check the satisfiability of

$$(\forall i \in [i_{\min}, i_{\max}])\ (\forall r \in [0.95\, r_0, 1.05\, r_0])\ (ri \leq u_{\max}). \quad (1)$$

Examples can be found in [2] where the safety condition to be verified is a minimal mutual distance between some satellites during a complete revolution.

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

In design problems, requirements are a bit more general. Instead of verifying theses conditions, we have to determine values of design variables that satisfy safety and performance conditions for all values of uncertain physical data. All these problems can be modeled in the framework of quantified constraint satisfaction problems (QCSPs). QCSPs arise naturally in numerous other applications (cf. [15, 16, 2, 12, 13, 23]).

In this paper, we are interested in a restricted form of QCSPs where existential quantifiers precede universal quantifiers, i.e.

$$\exists x \in \mathbf{x}, \forall y \in \mathbf{y}, c_1(x,y) \wedge \cdots \wedge c_p(x,y), \quad (2)$$

where $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_m)$ are vectors of variables, $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ and $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$ are vectors of intervals over continuous domains, and constraints c_i are inequalities of the form $f_i(x,y) \leq 0$. In addition to the problems introduced in [2], this class of QCSPs can tackle the design of robust controllers, which has been addressed in numerous works [10, 15, 9, 21, 28, 8].

In general, we are not only interested in the decision problem (2) but also in finding assignments of the existential variables which satisfy the constraints.

In other words, we consider $\mathbf{x}$ as a search space where we try to find values $x \in \mathbf{x}$ such that the relation $\forall y \in \mathbf{y}, c_1(x,y) \wedge \cdots \wedge c_p(x,y)$ holds. Such a value x is called a solution of (2) and we define the solution set of (2) in the following way:

$$\mathrm{Sol} := \{x \in \mathbf{x}\ :\ \forall y \in \mathbf{y}\ ,\ \bigwedge_{i \in \{1,\ldots,p\}} c_i(x,y)\}. \quad (3)$$

An essential observation is that in all mentioned applications, solution sets are continuums of solutions and that users are not really interested by isolated solutions but by continuous subsets of $\mathbf{x}$ where all points are solutions. Indeed, in design problems determining the real value of a physical value without any tolerance information does not really make sense. That's why we compute two sets $\mathcal{I}$ and $\mathcal{B}$ (called respectively the *inner approximation* and the *boundary approximation*) which satisfy the following relation:

$$\mathcal{I} \subseteq \mathrm{Sol} \subseteq \mathcal{I} \cup \mathcal{B}. \quad (4)$$

Set $\mathcal{I}$ contains only solutions whereas set $\mathcal{B}$ contains the boundary of the solution set. Thus $\mathcal{B}$ contains indifferently solutions and non-solutions. $\mathcal{I} \cup \mathcal{B}$ contains the whole solution set and is called the *outer approximation*. Actually, due to computational limitations, $\mathcal{I}$ and $\mathcal{B}$ are unions of boxes (cf. Figure 1).

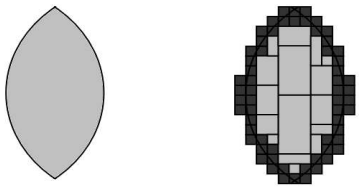

Figure 1: On the left hand side, the solution set Sol**; on the right hand side, the inner approximation $\mathcal{I}$ in light gray and the boundary approximation $\mathcal{B}$ in dark gray.**

In this paper, we propose a new algorithm for computing the sets of boxes $\mathcal{I}$ and $\mathcal{B}$. In contrast with other methods dedicated to the resolution of QCSPs, we handle the QCSP (2) as a non quantified CSP with quantified constraints: we introduce a CSP equivalent to (2) formed of constraints $c^{\mathbf{y}}(x)$ on the variables x (the domain parameter $\mathbf{y}$ is explicitly given in exponent for clarity). These constraints are quantified inequalities: $c^{\mathbf{y}}(x) \equiv \forall y \in \mathbf{y}, f(x,y) \leq 0$; y are called the parameters and $\mathbf{y}$ their domains. We propose here a generic branch and prune algorithm dedicated to continuous CSP with parametric constraints. This algorithm is based on a specific implementation for universally quantified inequalities of the following techniques:

– Pruning the non solution set;
– Identifying the solution set;
– Handling of parameter domains.

Other methods [1, 24, 2, 25] have been proposed to compute approximations of the solution set of (2). The new algorithm we propose here is both much simpler than the previously proposed algorithms, and also much more efficient. First experimentation show that our algorithm outperforms the methods introduced in [1, 24, 2, 25].

Notations. Boldface symbols denotes intervals, e.g. $\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} : \underline{x} \leq x \leq \overline{x}\}$ where $\underline{x}$ and $\overline{x}$ belong to a finite subset of $\mathbb{R}$ (usually the floating point numbers, cf. [11]). The set of these intervals is denoted by $\mathbb{IR}$ and the set of n dimensional interval vectors (also called boxes) by $\mathbb{IR}^n$. The width of an interval vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is $\text{wid}(\mathbf{x}) = \max_i |\underline{x}_i - \overline{x}_i|$ and its midpoint is $\text{mid}(\mathbf{x})$. $\tilde{x}$ denotes a value of $\mathbf{x}$. For two boxes $\mathbf{x}$ and $\mathbf{y}$, the interval hull is denoted by $\mathbf{x} \vee \mathbf{y}$ and is the smallest box which contains both $\mathbf{x}$ and $\mathbf{y}$ (note the difference between the union $[-1,1] \cup [2,3] = \{x \in \mathbb{R} : -1 \leq x \leq 1 \vee 2 \leq x \leq 3\}$ which is disconnected, and the interval hull $[-1,1] \cup [2,3] = [-1,3]$). Also, their set difference is denoted by $\mathbf{x} \backslash \mathbf{y} = \{x \in \mathbf{x} : x \notin \mathbf{y}\}$.

Outline of the paper. Section 2 recalls some basics on CSP with continuous domains. Section 3 describes the key features of the algorithm we propose. First experimental results are given in Section 4.

2. BASICS ON CSP WITH CONTINUOUS DOMAINS

To tackle CSP with continuous domains, a key issue is to be able to prove properties on continuums of real numbers. Interval analysis handles this problem in an efficient way using only computations on floating point numbers.

We recall here some basics which are required to understand the paper. More details can be found in [22, 17].

An interval contractor for a constraint c with n variables is a function

$$\text{Contract}_c : \mathbb{IR}^n \longrightarrow \mathbb{IR}^n, \tag{5}$$

that satisfies the following properties: (1) $\text{Contract}_c(\mathbf{x}) \subseteq \mathbf{x}$; (2) $\forall x \left(x \in \mathbf{x} \wedge c(x)\right) \Longrightarrow x \in \text{Contract}_c(\mathbf{x})$. Such contractors can be implemented using various techniques [22, 19, 4, 7] for standard inequality constraints. Examples and experimentation presented in this paper use contractors based on the $2B$–consistency.

$2B$–consistency [5, 19] (also known as hull consistency) states a local property on the bounds of the domain of a variable at a single constraint level. A constraint c is $2B$-consistent if, for any variable x_i, there exist values in the domains of all other variables that satisfy c when x_i is fixed to $\underline{x}_i$ and $\overline{x}_i$.

The filtering by $2B$–consistency of $P = (\mathbf{x}, \mathcal{C})$ is the CSP $P' = (\mathbf{x}', \mathcal{C})$ such that

- P and P' have the same solutions;
- P' is $2B$-consistent;
- $\mathbf{x}' \subseteq \mathbf{x}$ and the domains in $\mathbf{x}'$ are the largest ones for which P' is $2B$-consistent.

Filtering by $2B$–consistency of P always exists and is unique [19], that is to say, it is a closure.

3. DESCRIPTION OF THE ALGORITHM

The key idea of our algorithm is to reformulate a QCSP as a CSP with parametric constraints. More precisely, we reformulate a QCSP with constraints of type (2) as a CSP $\langle \mathcal{V}, \mathcal{C}, \mathcal{D} \rangle$ where:

– $\mathcal{V} = (x_1, \ldots, x_n)$;
– $\mathcal{C} = \{c_1^{\mathbf{y}}, \ldots, c_p^{\mathbf{y}}\}$ with $c_i^{\mathbf{y}}(x) \equiv \forall y \in \mathbf{y}, f_i(x,y) \leq 0$;
– $\mathcal{D} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

The parameter domains are considered at the level of each constraint, so we can process them differently for each constraint; they are initialized with $\mathbf{y}$. Standard techniques have to be adapted for this class of constraints. For example, we show in Subsection 3.2 that $\text{Contract}_{\forall y \in \mathbf{y}, f(x,y) \leq 0}(\mathbf{x})$ can be achieved with $\text{Contract}_{f(x,\tilde{y}) \leq 0}(\mathbf{x})$ for any $\tilde{y} \in \mathbf{y}$, the latter being implemented using standard interval contractors.

3.1 Outline of the Algorithm

The branch and prune algorithm we propose alternates pruning and branching steps in a standard way to reject parts of the search space that do not contain any solution. This process is interleaved with the identification of inner boxes that contain only solutions.

We also introduce new parameter instantiations and parameter reduction techniques which play a key role during the pruning of the search space and the identification of the solution sets. These key points are detailed in subsections 3.2, 3.3 and 3.4. They are implemented within a branching process (see Algorithm 1) using the following functions:

Function *ParameterInstantiation()* returns a set of constraints $\mathcal{C}'$ which is equivalent to $\mathcal{C}$ on the domain $\mathbf{x}$, but

Algorithm 1: Generic Branch and Prune Algorithm

Input: $\mathcal{C}$, $\mathbf{x}$, ϵ
Output: $(\mathcal{I}, \mathcal{B})$

$\mathcal{U} \leftarrow \{(\mathcal{C}, \mathbf{x})\}$; $\mathcal{B} \leftarrow \{\}$; $\mathcal{I} \leftarrow \{\}$;
while $\neg$empty($\mathcal{U}$) **do**
  $(\mathcal{C}, \mathbf{x}) \leftarrow$ extract($\mathcal{U}$);
  if wid($\mathbf{x}$) $> \epsilon$ **then**
    $\mathcal{C}' \leftarrow$ ParameterInstantiation($\mathcal{C}, \mathbf{x}$);
    $\mathbf{x}' \leftarrow$ Pruning($\mathcal{C}', \mathbf{x}$);
    $(\mathcal{C}'', \mathbf{x}'', \mathcal{I}') \leftarrow$ SolutionIndentification($\mathcal{C}', \mathbf{x}'$);
    $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{I}'$;
    $\mathcal{C}''' \leftarrow$ ParameterDomainBisection($\mathcal{C}'', \mathbf{x}''$);
    $\mathcal{U} \leftarrow \mathcal{U} \cup$ Branching($\mathcal{C}''', \mathbf{x}''$);
  else
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathbf{x}\}$;
  end
end
return $(\mathcal{I}, \mathcal{B})$;

where some parameters have been instantiated (see Subsubsection 3.4.3).

Function *Pruning()* contracts the domain $\mathbf{x}$ without loosing any solution of $\mathcal{C}'$ (see Subsection 3.2).

Function *SolutionIndentification()* contracts $\mathbf{x}'$ to $\mathbf{x}''$ and returns a list of inner boxes $\mathcal{I}'$ (see Subsection 3.3). The domains of the parameters are also updated in $\mathcal{C}''$ (see Subsubsection 3.4.1).

Function *ParameterDomainBisection()* bisects the parameter domains of the constraints and thus increases the number of constraints in $\mathcal{C}''$ (see Subsubsection 3.4.2).

Function *Branching()* achieves a standard bisection of the domains $\mathbf{x}''$.

3.2 Pruning

Local Pruning. Given a constraint $\forall y \in \mathbf{y}^i, c_i(x, y)$ and a box $\mathbf{x}$, we want to contract $\mathbf{x}$ rejecting only parts which do not contain any solution of this constraint. To achieve this task, Benhamou et al. and Ratschan [2, 25] apply $\text{Contract}_{c_i(x,\mathbf{y}^i)}(\mathbf{x})$, where y is handled as an existentially quantified variable in the domain $\mathbf{y}^i$. However, this strategy lacks efficiency. That's why we propose a better contractor by instantiating the parameter to an arbitrary value $\tilde{y} \in \mathbf{y}^i$ (see Example 1). More formally, the box $\mathbf{x}$ is contracted using $\text{Contract}_{c_i(x,\tilde{y})}(\mathbf{x})$ which rejects only parts of $\mathbf{x}$ that do not satisfy $c_i(x, \tilde{y})$, and thus, do not satisfy $\forall y \in \mathbf{y}^i, c_i(x, y)$.

Many strategies can be used to choose $\tilde{y}$. We chose $\tilde{y} = \text{mid}(\mathbf{y}^i)$. Experimentation have shown that searching for a better value of $\tilde{y}$ is not worthwhile. This is due to the fact that the parameter handling techniques reduce the impact of this choice (see examples 4 and 5 in Subsection 3.4).

Example 1. Let us consider the constraint $c(x)$ defined by $\forall y \in \mathbf{y}, f(x, y) \leq 0$ with $f(x, y) = 10y - x - y^2$, $\mathbf{y} = [0, 1]$, and $\mathbf{x} = [0, 15]$. The solution set of this simple CSP is the interval $[9, 15]$. To reject values of $\mathbf{x}$ that do not satisfy $c(x)$, we apply $\text{Contract}_{f(x,0.5)\leq 0}(\mathbf{x})$, which reduces $\mathbf{x}$ to $\mathbf{x}' = [4.75, 15]$. The method proposed in [2, 25] computes $\text{Contract}_{f(x,\mathbf{y})\leq 0}(\mathbf{x})$ but cannot achieve any contraction.

Global Pruning. The contraction of $\mathbf{x}$ using one constraint removes solutions of this constraint, and therefore solutions of the global CSP. This is illustrated in the first row of Figure 2 where diagrams (a) and (b) show two local contractions leading to $\mathbf{x}^{1\prime}$ and $\mathbf{x}^{2\prime}$. The global contraction depicted in Diagram (c) is obtained by computing $\mathbf{x}^{1\prime} \cap \mathbf{x}^{2\prime}$. Of course, in practice we compute this intersection in an incremental way.

3.3 Identification of Sets of Solutions

As in [6, 24, 26, 2, 25, 27], the identification of sets of solutions is implemented by applying interval contractors to the negation of the constraints. Again, this process is achieved for each constraint separately. However, now only areas that are proved to be local solutions for all constraints are solutions of the whole CSP.

Identification of Solutions for a Single Constraint. In order to identify parts of a box $\mathbf{x}$ which contains only solutions of the constraint $\forall y \in \mathbf{y}^i, f_i(x, y) \leq 0$, we compute

$$\langle \mathbf{x}^{i\prime}, \mathbf{y}^{i\prime} \rangle = \text{Contract}_{f_i(x,y)\geq 0}(\langle \mathbf{x}, \mathbf{y}^i \rangle), \tag{6}$$

where $\langle \mathbf{x}, \mathbf{y} \rangle$ stands for the vector $(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{y}_1, \ldots, \mathbf{y}_m)$. Thus, every value of $\mathbf{x} \backslash \mathbf{x}^{i\prime}$ satisfies $f_i(x, y) < 0$ for all values of y in $\mathbf{y}^i$. Indeed, the part of $\langle \mathbf{x}, \mathbf{y}^i \rangle$ which has been pruned does not contain any solution:

$$\forall x \in \mathbf{x}\, \forall y \in \mathbf{y}^i\ ,\ (x \notin \mathbf{x}^{i\prime} \vee y \notin \mathbf{y}^{i\prime}) \Longrightarrow \neg(f_i(x, y) \geq 0). \tag{7}$$

From (7) we have trivially

$$\forall x \in \mathbf{x}\, \forall y \in \mathbf{y}^i\ ,\ x \notin \mathbf{x}^{i\prime}\ \Longrightarrow\ f_i(x, y) < 0. \tag{8}$$

This process is illustrated on the following example.

Example 2. Continuing Example 1, let us consider $c(x)$ and $\mathbf{x} = [4.75, 15]$. In order to identify values of $\mathbf{x}$ that satisfy $c(x)$, we apply $\text{Contract}_{f(x,y)\geq 0}(\mathbf{x}, \mathbf{y})$ and obtain $\mathbf{x}' = [4.75, 10]$ and $\mathbf{y}' = [0.475, 1]$. Therefore, $\mathbf{x} \backslash \mathbf{x}' =]10, 15]$ contains only solutions of the constraint.

Identification of Solutions of the whole CSP. The $\mathbf{x}^{i\prime}$ for $i \in \{1, \ldots, p\}$ have been computed in such a way that all values of $\mathbf{x} \backslash \mathbf{x}^{i\prime}$ verify $c_i(x)$ (cf. the diagrams (d) and (e) of the second row of Figure 2). Thus, the values of $\mathbf{x}$ that are outside all $\mathbf{x}^{i\prime}$ satisfy all the constraints. Formally,

$$\mathbf{x} \backslash (\mathbf{x}^{1\prime} \cup \cdots \cup \mathbf{x}^{p\prime}) \tag{9}$$

contains only solutions of the CSP. This inner approximation (9) is the dashed area of Diagram (f) of the second row of Figure 2. Its description becomes very complicated for higher dimensions and when numerous constraints are involved. As in [25], we use instead the weaker inner approximation

$$\mathbf{x} \backslash (\mathbf{x}^{1\prime} \vee \cdots \vee \mathbf{x}^{p\prime}). \tag{10}$$

This inner approximation is represented on Diagram (g) of the second row of Figure 2. Let us define $\mathbf{x}' = \mathbf{x}^{1\prime} \vee \cdots \vee \mathbf{x}^{p\prime}$. The closure[1] of $\mathbf{x} \backslash \mathbf{x}'$ is added to the set of inner boxes while $\mathbf{x}'$ has to be further explored.

[1]The set difference $\mathbf{x} \backslash \mathbf{x}'$ is not closed in general, e.g. $[-10, 10] \backslash [-1, 1] = [-10, -1[\cup]1, 10]$. Nevertheless, we can observe that if f is continuous and non positive in $\mathbf{x} \backslash \mathbf{x}'$, then it is also non positive in its closure $\text{cl}(\mathbf{x} \backslash \mathbf{x}')$, e.g. if

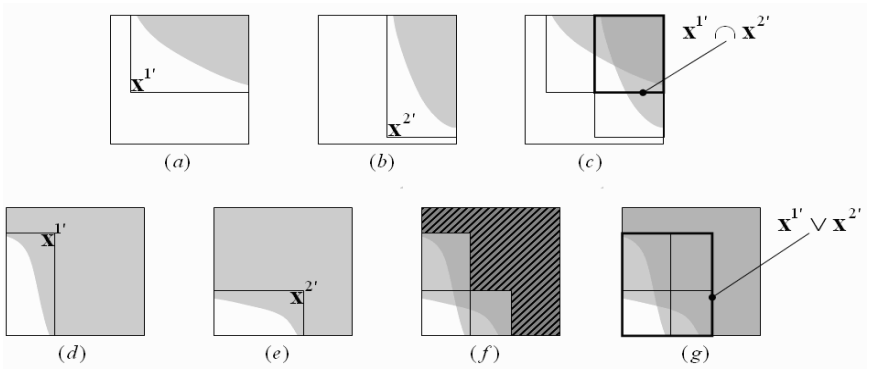

Figure 2: Upper row: pruning computed from local contractions. Lower row: identification of solutions using local contractions of the negation of the constraints.

3.4 Handling Parameter Domains

As said before, the size of the domains of the parameters is a critical issue for an efficient application of interval contractors. This section details the three methods implemented in our algorithm to overcome this problem. The goal of these methods is to only apply interval contractors to parts of the initial parameter domains while keeping the CSP solution set unchanged.

3.4.1 Parameter Domain Pruning

During the identification of solutions (6) the domains of the parameters $\mathbf{y}^i$ are reduced to $\mathbf{y}^{i\prime}$. In [2, 25], the initial parameter domain is restored, so this contraction of the parameter domains is not propagated. However, this reduced domain can be used while keeping the solution set of the constraint unchanged. Indeed, (7) trivially entails

$$\forall x \in \mathbf{x}\ \forall y \in (\mathbf{y}^i \backslash \mathbf{y}^{i\prime})\ f_i(x,y) < 0, \tag{11}$$

As a consequence, for every fixed x in $\mathbf{x}$, $\forall y \in \mathbf{y}^{i\prime}, c(x,y)$ implies $\forall y \in \mathbf{y}^i, c(x,y)$. Finally, we can replace the latter by the former keeping the solution set unchanged during the solution identification step of the algorithm. This is a significant improvement that is implemented without any overhead since the parameter domains have to be contracted anyway during the solution identification step.

Example 3. In Example 2, the identification of solutions step has contracted the parameter domain from $\mathbf{y} = [0,1]$ to $\mathbf{y}' = [0.475, 1]$. This latter domain can be used in future processing while keeping unchanged the CSP solution set.

3.4.2 Parameter Domain Bisection

Bisecting parameter domains is the most obvious way to reduce them. This is a critical issue to improve the convergence of the algorithm. Bisection is also used in [25] but implemented in a different way. Here, we change the constraint $\forall y \in \mathbf{y}, c(x,y)$ to the conjunction of the two constraints $\forall y \in \mathbf{y}^1, c(x,y)$ and $\forall y \in \mathbf{y}^2, c(x,y)$, where $\mathbf{y}^1$ and $\mathbf{y}^2$ are obtained by bisecting $\mathbf{y}$. The following example illustrates how the parameter domain bisection is performed, and how it improves the pruning process as well as the identification of solutions.

Example 4. Let us come back to Example 1 where the constraint store contains only one constraint $\mathcal{C} = \{(\forall y \in \mathbf{y}, f(x,y) \leq 0)\}$. Bisecting parameter domains, we obtain the new constraint store $\mathcal{C}' = \{(\forall y \in \mathbf{y}^1, f(x,y) \leq 0), (\forall y \in \mathbf{y}^2, f(x,y) \leq 0)\}$ with $\mathbf{y}^1 = [0, 0.5]$ and $\mathbf{y}^2 = [0.5, 1]$. The latter constraint store is equivalent to the original one, but contains more constraints with smaller parameter domains.

The pruning operator is now applied to each constraint of the store, thus the two contractions $\text{Contract}_{f(x,0.25)\leq 0}(\mathbf{x})$ and $\text{Contract}_{f(x,0.75)\leq 0}(\mathbf{x})$ are computed. This leads to $\mathbf{x}' = [6.9375, 15]$. The pruning achieved here is sharper than the one computed without bisecting the parameter domains (cf. Example 1).

To identify solutions of this new CSP, we compute $(\mathbf{x}^{i\prime}, \mathbf{y}^{i\prime}) = \text{Contract}_{f(x,y)\geq 0}(\mathbf{x}', \mathbf{y}^i)$ for $i \in \{1,2\}$. We obtain $\mathbf{x}^{1\prime} = \emptyset$, $\mathbf{y}^{1\prime} = \emptyset$, $\mathbf{x}^{2\prime} = [6.9375, 9.75]$ and $\mathbf{y}^{2\prime} = [0.71875, 1]$. Finally, $\mathbf{x}'$ is contracted to $\mathbf{x}^{1\prime} \vee \mathbf{x}^{2\prime} = [6.9375, 9.75]$ while $\text{cl}([6.9375, 15] \backslash [6.9375, 9.75]) = [9.75, 15]$ is proved to be an inner box. The solution identification is also more efficient thanks to the bisection of the parameter domain (cf. Example 2).

3.4.3 Parameter Instantiation

The constraint $\forall y \in \mathbf{y}, f(x,y) \leq 0$, where $\mathbf{y} = [\underline{y}, \overline{y}]$, can be simplified if the function f is proved to be monotonic w.r.t. the parameter. Indeed, if f is increasing (resp. decreasing) w.r.t. y, then the quantified constraint is obviously equivalent to the non-quantified constraint $f(x, \overline{y}) \leq 0$ (resp. $f(x, \underline{y}) \leq 0$). The variation of the function can be checked by evaluating its derivative w.r.t. the parameter on the intervals $\mathbf{x}$ and $\mathbf{y}$. The same property holds if there are several parameters, using a derivative w.r.t. each parameter.

Example 5. Let us come back to Example 1. Evaluating

$$(\partial f/\partial y)(\mathbf{x}, \mathbf{y}) = 10 - 2\mathbf{y} = [8, 10], \tag{12}$$

we prove that f is increasing w.r.t. y. Therefore, the constraint $\forall y \in \mathbf{y}, f(x,y) \leq 0$ is equivalent to $f(x,1) \leq 0$. Then, the pruning contracts $\mathbf{x} = [0, 15]$ to $[9, 15]$ and the solution identification proves that $[9, 15]$ contains only solutions. On this simple example, we obtain an exact description of the solution set. This set is much sharper than the contractions obtained without this instantiation of the parameter (cf. examples 1 and 2).

The parameter instantiation can drastically improve the efficiency of the pruning and the solution identification steps.

$f(x) \leq 0$ in $[-10, -1[\cup]1, 10]$ then $f(x) \leq 0$ also holds in $[-10, -1] \cup [1, 10]$. Thus, $\text{cl}(\mathbf{x} \backslash \mathbf{x}')$, which can be described by a simple union of boxes, is recorded instead of $\mathbf{x} \backslash \mathbf{x}'$.

			Qine	
problem	ratio	Rsolver	2B	2B+
Circle	0.98	55.67	0.90	1.03
	0.99	169.51	2.32	2.22
	0.999	–	69.18	31.55
PathPoint	0.6	122.72	0.01	0.01
	0.65	334.34	0.01	0.02
	0.7	–	0.02	0.03
Parabola	0.96	48.33	1.47	1.36
	0.97	130.30	2.74	1.92
	0.98	–	8.77	6.59
Robot	0.98	10.34	0.06	0.08
	0.99	26.74	0.14	0.18
	0.999	–	5.37	4.08
Satellite	0.5	303.30	71.36	114.33
	0.55	–	168.32	268.36
	0.6	–	227.09	368.90
Robust1	0.999	1.10	0.01	0.00
	0.9999	7.16	0.04	0.00
	0.99999	76.25	0.10	0.00
Robust4	0.5	–	$-^*$	0.00
	0.55	–	–	0.01
	0.6	–	–	0.01
Robust5	0.7	75.35	0.00	0.00
	0.75	80.62	0.00	0.00
	0.8	–	0.01	0.00
Robust6	0.7	59.98	0.02	0.00
	0.75	224.33	0.05	0.00
	0.8	–	0.09	0.00

Table 1: Timing (in seconds) for Rsolve vs Qine

However, evaluating derivatives can be expensive, in particular when they involve trigonometric function. Experimentation presented in the next section illustrate well this trade-off.

4. EXPERIMENTATION

This section compares the results of the IPA system described in [2], Rsolver [25] from S. Ratschan with our system, called "Qine", on a set of 9 benches.

The 9 benches come from the literature. The Circle, PathPoint, Parabola, Robot and Satellite QCSPs are taken from [2]. A description of Robust1 (respectively, Robust4, Robust5 and Robust6) can be found in [8] (respectively, [21], [15] and [14]).

The Rsolver and Qine benches have been run on an Intel Core Duo 2 at 2.4Ghz with a time out of 600s. To limit the effect of the high memory consumption of these algorithms, the available memory has been restricted to 1Gb. Thus, a bench could either succeed to run within these two limits, end with a time out ("–"), or reach the memory limit ("$-^*$").

Table 1 reports the results obtained with Rsolver and Qine. It gives Rsolver timing, as well as the time required to solve the benches for the different Qine running options :

- "2B" uses the contractor based on 2*b*-consistency techniques. More precisely, its implementation relies on a forward-backward evaluation of the direct acyclic graph which represents the constraint.
- "2B+" combines the previous contractor with the derivative based parameter handling strategy introduced in subsection 3.4.3.

		IPA		QINE	
problem	ratio	t	$t * corr$	2B	2B+
Circle	0.8928	2.4	1.402	0.14	0.20
	0.9326	9.328	5.450	0.20	0.30
	0.9535	64.876	37.907	0.30	0.42
PathPoint	0.8172	148.66	86.86	0.08	0.12
Parabola	0.8716	0.340	0.1986	0.06	0.06
	0.9340	2.824	1.65	0.34	0.33
	0.9650	75.936	44.3697	1.80	1.65
Robot	0.9924	1.112	0.6497	0.26	0.34
	0.9973	5.908	3.452	1.07	1.10
	0.9980	16.933	9.894	1.89	1.81
Satellite	0.2813	5.660	3.3071	18.41	29.61
	0.4844	27.633	16.146	52.79	84.90

Table 2: Timing (in seconds) for IPA vs Qine

The results have been computed according to a ratio (column 2) where

$$ratio = \frac{V_{inner} + V_{outer}}{V_{initial}} \tag{13}$$

where V_{inner} is the total volume of the inner boxes, V_{outer} is the total volume of the outer boxes and $V_{initial}$ is the volume given by the initial domains of the variables. Though all the benches have been run for ratios going from 0.5 to 0.99999, the table gives only the most significant results for the sake of space.

As shown in table 1, in average, Qine outperforms Rsolver by one order of magnitude. For instance, Qine handles the Robust benches immediately while Rsolver needs much for time to do so. A comparison of the different available combinations shows that the "2B+" combination has a more robust behavior. It takes advantage of all the available information and offers a good trade-off between the computation time and the domain reductions. However, on the Satellite bench, 2B performs better than 2B+. This illustrates the trade-off between the cost of derivative computation and the benefit of parameter instantiations.

Tables 2 compares IPA with Qine. IPA system has been run on a Pentium M at 2Ghz running Linux. To allow a fair comparison, we have computed a timing correction : the same system, Rsolver, has been run on both systems in order to determine this correction. Therefore, the initial timing obtained for IPA (column 3) has been multiplied by 0.58 (column 4) to allow a fair comparison between the two systems.

Here again, Qine outperforms IPA. Indeed, IPA was not able to solve the PathPoint bench within the timeout for only one of the tested ratios. However, IPA is faster than Qine on the Satellite bench for the lower ratio values. This behavior is probably due to the limit of the 2B based contractor whose domain reduction capabilities decrease when a variable has multiple occurrences within one constraint. IPA is based on a Box contractor which does not suffer from the same behavior (see [7] for a detailed comparison of 2B and Box). However, when the ratio value increases, Qine becomes faster than IPA. For instance, IPA needs more than 876s (corrected time) to solve the Satellite bench for a ratio of 0.6276 whereas Qine achieves this task with 2B within 401.44s.

Note that the class of QCSPs handled by IPA is limited to one parameter. Thus, IPA is not able to solve the Robust

benches.

5. CONCLUSION

In this paper, we have introduced a new, simple and efficient algorithm to handle a significant class of QCSPs. Examples coming from the literature reveal that this class covers most of the practical applications.

Our algorithm is based on new techniques for handling parameters. It also takes advantage of information provided by the derivatives to improve the contraction of the domains involved in constraints with parameters, a key issue in the efficient solving of QCSPs.

Experimentation underline the efficiency of our algorithm which outperforms two of the available state of the art implementations able to handle such QCSPs. In average, our implementation is by one order of magnitude quicker than the two other systems.

Further work concerns the improvement of the implementation of the contractor by using the best filtering techniques for each type of constraints and the generalization of the use of available information to still enhance the speed of the solving process.

Acknowledgments

We are grateful to Marc Christie for his valuable help in the experimentation.

6. REFERENCES

[1] F. Benhamou and F. Goualard. Universally quantified interval constraints. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, volume 1894 of *LNCS*, pages 67–82, 2000.

[2] F. Benhamou, F. Goualard, E. Languenou, and M. Christie. Interval Constraint Solving for Camera Control and Motion Planning. *ACM Trans. Comput. Logic*, 5(4):732–767, 2004.

[3] F. Benhamou and W. Older. Applying interval arithmetic to real, integer and boolean constraints. *Journal of Logic Programming*, pages 1–24, 1997.

[4] F. Benhamou, D. A. McAllester, and P. V. Hentenryck. CLP(Intervals) Revisited. In *SLP*, pages 124–138, 1994.

[5] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, pages 125–149, 1987.

[6] H. Collavizza, F. Delobel, and M. Rueher. Extending Consistent Domains of Numeric CSP. In *Proceedings of IJCAI 1999*.

[7] H. Collavizza, F. Delobel, and M. Rueher. Comparing Partial Consistencies. *Reliab. Comp.*, 1:1–16, 1999.

[8] P. Dorato. Quantified multivariate polynomial inequalities. *IEEE Control Systems Magazine*, 20(5):48–58, 2000.

[9] P. Dorato, W. Yang, and C. Abdallah. Robust multi-objective feedback design by quanti er elimination. *J. of Symbolic Computations*, 24:153–159, 1997.

[10] G. Fiorio, S. Malan, M. Milanese, and M. Taragna. Robust performance design of fixed structure controllers for systems with uncertain parameters. In *Proceedings of the 32st IEEE Conference on Decision and Control*, volume 4, pages 3029 – 3031, 1993.

[11] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *Computing Surveys*, 23(1):5–48, 1991.

[12] A. Goldsztejn. A Branch and Prune Algorithm for the Approximation of Non-Linear AE-Solution Sets. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1650–1654.

[13] A. Goldsztejn and L. Jaulin. Inner and Outer Approximations of Existentially Quantified Equality Constraints. In *Proceedings of CP 2006*, volume 4204/2006 of *LNCS*, pages 198–212.

[14] L. Jaulin, I. Braems, and E. Walter. Interval methods for nonlinear identification and robust control. In *In Proceedings of the 41st IEEE Conference on Decision and Control*, volume 4, pages 4676 – 4681, 2002.

[15] L. Jaulin and E. Walter. Guaranteed tuning, with application to robust control and motion planning. *Automomatica*, 32(8):1217–1221, 1996.

[16] M. Jirstand. Nonlinear control system design by quantifier elimination. *Journal of Symbolic Computation*, 24(2):137–152, 1997.

[17] R. B. Kearfott. Interval Computations: Introduction, Uses, and Resources. *Euromath*, Bulletin 2(1):95–112, 1996.

[18] O. Knueppel. PROFIL/BIAS - A Fast Interval Library. *Computing*, 53(3-4):277–287, 1994.

[19] O. Lhomme. Consistency Techniques for Numeric CSPs. In *Proceedings of IJCAI 1993*, pages 232–238.

[20] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, pages 99–118, 1977.

[21] S. Malan, M. Milanese, and M. Taragna. Robust analysis and design of control systems using interval arithmetic. *Automatica*, 33(7):1363–1372, 1997.

[22] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge Univ. Press, Cambridge, 1990.

[23] S. Ratschan. Applications of Quantified Constraint Solving over the Reals Bibliography. http://www.cs.cas.cz/ ratschan/appqcs.html.

[24] S. Ratschan. Approximate quantified constraint solving by cylindrical box decomposition. *Reliab. Comp.*, 8(1):21–42, 2002.

[25] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[26] X. Vu, D. Sam-Haroud, and M.-C. Silaghi. Approximation techniques for non-linear problems with continuum of solutions. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*, volume 2371 of *LNAI*, pages 224–241. Springer-Verlag, 2002.

[27] X.-H. Vu, M. Silaghi, D. Sam-Haroud, and B. Faltings. Branch-and-prune search strategies for numerical constraint solving. Technical Report LIA-REPORT-2006-007, Swiss Federal Institute of Technology (EPFL), 2006.

[28] M. Zettler and J. Garloff. Robustness analysis of polynomials with polynomial parameterdependency using bernstein expansion. *IEEE Transactions on Automatic Control*, 43(3):425–431, 1998.